Lecture 18: Online Algorithms, Prediction with Expert Advice

Objectives of this lecture

- Introduce concept of computational model, formalize online algorithms.
- Rent or buy problem.
- Prediction with expert advice, introduce multiplicative weights.

1 Philosophy

Last class we discussed strategies for zero-sum games. Such strategies, more generally, belong to the class of online algorithms.

In online algorithms, the input arrives over time rather than being known entirely up front. At each point in time, we have to make some decision, and each such decision is irrevocable, i.e., we can not change our mind later. Depending on the choices we make, we incur some cost, depending on the cost model of the problem. The goal is to perform well relative to an optimal *omniscient* algorithm, i.e., one that can predict the future and see the entire input in advance. This relative performance is captured by the notion of competitive ratio below.

Definition: Competitive Ratio

An online algorithm is called c*-competitive* if for all possible inputs σ

$$\mathrm{ALG}(\sigma) \leq c \cdot \mathrm{OPT}(\sigma),$$

where $\mathrm{ALG}(\sigma)$ is the cost incurred by the online algorithm on the input σ (which it does not know in advance) and $\mathrm{OPT}(\sigma)$ is the cost of an optimal omnipotent algorithm that can see σ in advance. The factor c is called the *competitive ratio* of the algorithm.

Online algorithm is a model of computation, which formalizes the operations that an algorithm can perform. One advantage of having a more restricted computational model (decisions are retrovocable) is a systematic approach for lower bounding the best competitive ratio possible. The inputs can be viewed as adversarial, aka. a player who plays against the algorithm with the goal of making the competitive ratio as large as possible.

2 Rent or buy?

Here is a simple online problem that captures a common issue in online decision-making, called the rent-or-buy problem.

Problem: Rent or buy

Its the middle of the snow season and you are planning on going skiing. You can rent a pair of skis at $\$r$ per day, or buy a pair for $\$b$ and keep them forever. You would like to ski for as many days as possible, however, you do not know how many more days of the season will be viable weather for skiing. Each day you find out whether the weather is still good. At some point, you discover that the ski season is over. Your choice is to decide whether to rent or buy skis on each day, with the goal of minimizing the total amount of money that you spend.

Let's walk through a concrete example. You can either rent skis for \$50 or buy them for \$500. If we know the future in advance, the solution is to buy immediately if we know that there are at least 10 days of viable skiing weather, and if not, just always rent. The tricky part is designing an *online* algorithm that doesn't know the future. It has no idea how many days of viable whether there will be. Lets start with some simple but sub-optimal strategies to illustrate:

- **Always immediately buy:** One valid online strategy is to immediate buy on the first day if the weather is good. The worst case input for the this strategy is when we only get to go skiing once, so we could have just paid \$50, so the competitive ratio is $500/50 = 10$, we paid $10\times$ more than we could have.

  Rent forever: Another strategy is to never buy skis and just always rent. In this strategy, the worst case input is that the ski season goes on arbitrarily long, and we end up paying an arbitrarily high amount of money, when the optimal choice would have been to buy immediately, so the competitive ratio is actually ∞ (or unbounded).

In general, since after buying the skis the algorithm has no more decisions to make, we can characterize any online algorithm for the rent-or-buy problem by the day on which it decides to buy. Now observe that in general, the worst-case input for such an algorithm is that the weather is bad on the day after it buys the skis. With this in mind, here is one more bad strategy before we hone in on the optimal one.

- **Rent five times then buy:** How about we rent five times, then decide that it is time to buy. The worst-case input is that the weather is good for six days, then bad. In this case, our algorithm pays $5 \times 50 + 500 = 750$, but the optimal algorithm would just always rent, which costs $6 \times 50 = 300$, so this is 2.5-competitive.

Well that's certainly a lot better than 10. It seems like if we hedge our bets by renting longer, we get a better competitive ratio. At some point, this will stop being true, though. In particular, it never makes sense to plan to rent for more than 10 days, because then we should have just bought the skis for sure. So the most hedging we can do is to rent for 10 days then buy. This is called the *better-late-than-never* algorithm.

Algorithm: Better-late-than-never

We rent for $b/r-1$ days[a], then we buy. In other words, we buy on day b/r.

[a] If r does not divide b, then we should rent for $\lfloor b/r \rfloor - 1$ days, but we will just assume that r divides b for simplicity

Theorem: Better-late-than-never is 2-competitive

Better-late-than-never is a 2-competitive algorithm for the rent-or-buy problem.

Proof. Suppose the weather is good for n days. We have to consider two cases:

1. If $nr < b$ (i.e., $n < b/r$), then the optimal solution is to always rent, but in this case, our algorithm doesn't buy either, so it is optimal.
2. If $nr \geq b$, the optimal solution buys immediately, but our algorithm first rents for $b/r-1$ days before buying, so the ratio is

$$\frac{\left(\frac{b}{r}-1\right)r+b}{b} = \frac{b-r+b}{b} = 2-\frac{r}{b} \leq 2.$$

□

3 Prediction with Expert Advice

We often deal with decisions more complicated than "recent" or "buy". So we can abstract them even further by defining predictions and outcomes.

There are n "experts". Each day the following sequence of events happens:

- We see the n experts' predictions of the outcome.
- We make our own prediction about the outcome.
- The actual outcome is revealed.
- We are correct if we made the right prediction, and make a mistake otherwise.

This process goes on indefinitely. Our goal: at any time (say after some T days), we want to have made not too many more mistakes than the best expert. We want to bound the number of mistakes, and hence this is called the "mistake-bound" model.

For the rest of this lecture, we only work with binary outcomes (e.g., "Up/Down" predictions if the stock market will go up or down, or "snow/spring" if the weather will be good or not.)

3.1 Warmup: Simple Strategies

Majority and halving Suppose we know the best expert makes no mistakes. Can we hope to make only a few mistakes? Here's a strategy that makes only $\lfloor \log_2 n \rfloor$ mistakes. Just predict what the majority of the remaining experts predicts. (In case of a tie, choose arbitrarily.) Take all

the experts that are wrong and discard them. So each time we make a mistake, we reduce by the number of experts by at least 1/2. This means after $\log_2 n$ mistakes we will be left with the perfect expert.

Without a perfect expert Suppose the best expert makes at most M mistakes on some sequence. Can we hope to make only a few mistakes? Here's a strategy that makes at most $(M+1)(\log_2 n + 1)$ mistakes, assuming n is a power of 2. Run the above majority-and-halving strategy, but when you have discarded all the experts, bring them all back (call this the beginning of a new "phase"), and continue.

Note that in each phase each expert makes at least one mistake, and you made at most $\log_2 n + 1$ mistakes. Hence, if the best expert makes only M mistakes, there would be at most M finished phases (plus the last unfinished one), and hence at most $(M+1)(\log_2 n + 1)$ mistakes in all.

3.2 The Multiplicative Weights Algorithm

Throwing away an expert when it makes a mistake seems too drastic. Suppose we instead assign weights w_j to the experts, sum the weights of the expert saying `Up`, sum the weights of the of the expert saying `Down`, and predict the outcome with greater weight. (This is called the *weighted majority* rule, since we are following the advice of the experts that form the weighted majority.)

Then once we see the outcome, we can reduce the weight of the experts who were wrong. In the above algorithm, we were zeroing out the weight, but suppose we are gentler?

The *(basic) deterministic weighted majority* algorithm does the following:

Algorithm: Deterministic weighted majority

Start with each expert having weight 1. Each time an expert makes a mistake, half its weight. Output the prediction of the experts who form the weighted majority.

Remarkably, we can get a much stronger result now.

Theorem 1

If on some sequence of days, the best expert makes M mistakes. The basic deterministic weighted majority algorithm makes $\leq 2.41(M + \log_2 n)$ mistakes.

Proof. Let $\Phi := \sum_{i=1}^{n} w_i$ be the sum of weights of the n experts. Note that initially $\Phi = n$. Moreover, we claim that each time we make a mistake

$$\Phi_{\text{new}} \leq \frac{3}{4}\Phi_{\text{old}}.$$

Indeed, at least half the weight (which was making the majority prediction) gets halved (because it made a mistake), so we lose at least a quarter of the weight with each mistake. (Also if

we don't make a mistake, Φ does not increase.) So if we've made m mistakes at some point, the total weight is at most

$$\Phi_{\text{final}} \leq (3/4)^m \cdot \Phi_{\text{init}} = (3/4)^m \cdot n.$$

Moreover, if the best expert i^* has made M mistakes, then $\Phi_{\text{final}} \geq w_{i^*} = (1/2)^M$. So

$$(1/2)^M \leq (3/4)^m \cdot n \quad \Rightarrow \quad (4/3)^m \leq n2^M.$$

Taking logs (base 2) and noting that $\frac{1}{\log_2(4/3)} = 2.41\ldots$ completes the proof. □

This is pretty cool! If the best expert makes mistakes 10% of the time we are wrong about 24% of the time (plus $O(\log n)$, but since this depends only on the number of experts, it will be a negligible fraction as $M \gg \log n$). We can improve the 2.41 factor down to as close to 2 as we want, which you can try to prove as an exercise.

Remark: Lower bound for deterministic experts

You can show that no deterministic prediction algorithm can make fewer than $2M$ mistakes. How? Two experts: one says "Up" all the time, the other "Down" all the time. Fix any prediction algorithm. Since this algorithm is deterministic, you (as the adversary) know what prediction it will make on any day, if you know what happened on all previous days. So as the adversary, you can now make the "real" outcome on this day be the opposite of the algorithm's prediction for this day. This means the algorithm makes a mistake on all days. But one of the experts must be right on at least 50% of the days.